Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tilen Matkovič

Spreminjanje velikosti slik glede na vsebino

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Luka Šajn

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Spreminjanje velikosti slik glede na vsebino

Tematika naloge:

Napišite program, ki zmanjša vhodno sliko na velikost, ki jo določi uporabnik oz. je omejena z napravo, ki sliko prikazuje (mobitel, tablica, zaslon ...). Izhodna slika naj vsebuje samo pomembne objekte. Program naj omogoča delovanje pri dveh načinih izreza slike: pri ročnem izrezu, kjer sami označimo, kateri del je na sliki pomemben in ga je treba ustrezno obrezati, in pri avtomatskem izrezu, kjer mora program sam ugotoviti, kaj izrezati. Tu si lahko pomagate z OCR, detekcijo obrazov in podobnimi algoritmi.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tilen Matkovič sem avtor diplomskega dela z naslovom:

Spreminjanje velikosti slik glede na vsebino

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luke Šajna,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 31. avgusta 2015 Podpis avtorja:

Moje zahvale gredo mentorju doc. dr. Luki Šajnu za napotke in pomoč pri izdelavi tega diplomskega dela in družini za podporo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
OCR	optical character recognition	optično prepoznavanje znakov
RGB	red green blue	rdeča zelena modra
SVD	singular value decomposition	singularni razcep
YCbCr	luminance; chroma: blue; chroma: red	svetlost; modra; rdeča

Povzetek

Cilj diplomskega dela je realizirati in napisati program, ki vhodni sliki spremeni velikost, tako da se čimbolj ohranijo pomembni objekti, ter opisati nekaj algoritmov, s katerimi se to lahko doseže.

V uvodu je opisan problem, ki ga v diplomi rešujemo. Drugi del se nanaša na nekaj možnih rešitev, s katerimi lahko rešimo naš problem. To so OCR, zaznavanje obrazov, ločevanje ključnih značilnic in rezanje šivov. Vsak od teh je podrobno opisan. V naslednjem poglavju je opisan pristop k problemu. Začnemo z opisom orodij, ki smo jih uporabili pri izdelavi programa, in algoritmom, ki smo ga realizirali kot mešanico algoritmov, opisanih v drugem poglavju (ločevanje ključnih značilnic in rezanje šivov). Četrti del vsebuje primere delovanja našega programa. Rezultati so slike, ki vsebujejo samo ključne oz. pomembne objekte. Pri naslednjem delu je opisano delovanje zaznavanja obrazov, ki bi lahko prišlo v poštev k našemu algoritmu. Na samem koncu so opisani sklep, ugotovitve in možnosti nadgradnje algoritma.

Ključne besede: OpenCV, računalniški vid, rezanje šivov, OCR, ločevanje značilnic, spreminjanje razmerja slik.

Abstract

The aim of the thesis is to realize and write a program that resizes the input image to preserve important objects and describe some algorithms by which this can be achieved.

The introduction describes the problem. The second part relates to some possible solutions that could solve the problem. These are OCR, face detection, salient feature extraction and seam carving. Each of these is described in detail. The next chapter describes the approach to the problem. We begin with descriptions of the tools that were used in the production of the algorithm, which was realized as a mixture of algorithms described in the second chapter (salient feature extraction and seam carving). The fourth part contains examples of how our program works. The results are images that only contain the important objects. The next section describes the operation of face detection, which we could realize in our algorithm. At the very end of the thesis we described the findings and possible upgrades of the algorithm.

Keywords: OpenCV, computer vision, seam carving, OCR, feature extraction, image resizing.

Poglavje 1

Uvod

Vsak dan se srečujemo s slikami, pa naj bodo velike ali majhne. Pri velikih slikah lahko pride do težave, da zavzamejo veliko prostora (na disku in na mediju – časopisu, ekranu, mobitelu ...). Idealno bi bilo, da bi obstajal kakšen program ali algoritem, ki bi, ne glede na velikosti slik, zmanjšal njihove dimenzije na poljubno velikost brez izgube ali deformacije glavnega objekta.

Do te ideje so med drugimi prišli tudi pri podjetju Zemanta, kjer so dali izziv študentom napisati program, ki bi reševal ta problem. K izzivu me je motiviral mentor, ki mi ga je predlagal in mi podal nekaj napotkov za reševanje.

Cilj diplomskega dela je izdelati program, ki zmanjša dimenzije slik, in predelati nekaj algoritmov, ki naj bi čim bolje opravili to delo. Sem spadajo OCR, detekcija obrazov, rezanje šivov, segmentacija slik, ločevanje značilnic... Program bi lahko prišel v poštev različnim podjetjem, ki se ukvarjajo z mobilnimi spletnimi aplikacijami. Če na primer neka slika presega dimenzije ekrana ciljne naprave, bi lahko s tem programom odrezali nepotrebne dele.

1.1 Predstavitev problema

Vsako sliko lahko iz prvotnih dimenzij zmanjšamo na izhodne dimenzije s skaliranjem ali pa izrezovanjem/rezanjem. Skaliranje je vrsta transformacije, pri kateri se višina in širina množita s skalarjem. Če je skalar med 0 in 1, gre za pomanjšavo, če pa je večji od 1, gre za povečavo slike. Poznamo uniformno in neuniformno skaliranje. Pri prvem so skalarji za vse koordinatne osi enaki, pri drugem pa različni. V našem primeru skaliranje ni dopustno, saj lahko izgubimo del informacij, zato lahko problem rešujemo z izrezovanjem.

Izziv je sestavljen iz dveh delov, in sicer iz ročnega in avtomatskega izreza iz slike. Pri Zemanti so implementacije ocenjevali glede na izvirnost, enostavnost, pravilnost, hitrost delovanja in zanimivost pristopa k problemu [6].

1.1.1 Ročni izrez

Pri ročnem izrezu že vnaprej določimo, kateri del slike je pomembnejši od drugih, in ga ne smemo odrezati. Druge dele slike lahko poljubno odstranimo, npr. zgoraj več, spodaj manj, vendar niso vsi načini enako primerni, zato je treba najti takega, ki bo čim boljše opravljal svoje delo.

1.1.2 Avtomatski izrez

V tem primeru mora program sam ugotoviti, kaj je pomembnejše od drugih delov slike, in spet paziti, da se teh delov ne odstrani oz. se jih odstrani čim manj. Tukaj pridejo v poštev razni algoritmi za detekcijo obrazov, besedila, rezanje šivov in podobni. Najboljši način je verjetno mešanica dveh ali več algoritmov. To lahko ugotovimo že iz samih algoritmov. Na primer eni prepoznavajo besedilo (OCR), drugi pa zaznavajo obraze. In če bi hoteli na sliki označiti besedilo in obraze, moramo uporabiti oba algoritma.

Poglavje 2

Manipulacija s slikami in iskanje pomembnih delov

V nadaljevanju je predstavljenih nekaj algoritmov, ki so se zdeli primerni za rešitev problema. Pri implementaciji sem jih zaradi več dejavnikov izbral nekaj, ki so se zdeli najboljši.

2.1 OCR

OCR ali ang. *'Optical Character Recognition'* je način pridobivanja besedila, pri katerem se človeku berljiv tekst pretvori v besedilo, ki je berljivo računalniku. OCR, ki se je včasih uporabljal predvsem pri skenerjih, se v današnjih časih uporablja tudi v mobilnih napravah, npr. pri zajemu slike. Eden bolj znanih je Googlov odprtokodni Tesseract, ki se uporablja tudi pri Androidu [15].

Prvi pravi začetki razvoja OCR segajo v 1940 s pojavitvijo digitalnega računalnika. V komercialne namene se je uporabljal šele v sredini petdesetih let [12]. Potem je sledila prva generacija OCR, ki je bila zelo omejena. Pri drugi generaciji so lahko brali tudi ročno pisane črke in številke. Izziv pri tretji generaciji (sredina sedemdesetih do sredina osemdesetih let) so bili dokumenti slabe kvalitete. Predvsem zaradi visokih cen so bili aparati OCR

Slika 2.1: Normalizacija in zameglitev znaka [12]

zelo redki. Pocenitev strojne in druge opreme je nastopila po letu 1986.

Glavni simboli pri OCR so male in velike črke, številke in posebni znaki, kot so na primer vejice, pike in druga ločila. Upoštevati je treba tudi več slogov, ki jih lahko spreminjamo pri besedilu. Postopek se začne z optičnim skeniranjem oz. danes lahko že s samim zajemom slike iz fotoaparata. Pri tem se uporablja zmanjšanje barv v sivinske in upragovanje (opisano v 2.3.1). Sledita lokalizacija in segmentacija. Pri lokalizaciji je treba locirati regije, kjer sploh so besedila, in jih ločiti od slik oz. druge grafike. Večina algoritmov OCR segmentira vsako črko posebej. Segmentacija se izvaja z izolacijo črnih neprekinjenih komponent. Težave nastopijo, če so znaki sestavljeni iz več delov. Pojavljajo se tudi težave, kot so na primer [12]:

- prekrivanje črk;
- pikice (rezultat šuma pri upragovanju) se zamenjajo za črke;
- grafični objekti se zamenjajo za tekst in obratno.

Po tem sledi predprocesiranje. Tu se delno popravljajo napake, ki so posledica prejšnjih postopkov. Sem spadata normalizacija in zameglitev znakov (glej sliko 2.1).

Zatem se je treba spopasti z najtežjim delom, in sicer ločevanjem značilnic. Za boljšo robustnost se mora upoštevati naslednjih 5 kriterijev [12]:

- šum;
- popačenost;

- slog;
- translacija;
- rotacija.

Tu je mnogo različnih pristopov. Pri enostavnejših se ne gleda nobenih značilnosti znakov, ampak se takoj preveri s predlaganimi znaki. Zahtevnejši pristopi uporabljajo razne meritve in analize. Od tod naprej se izvaja klasifikacija znakov. To je proces klasifikacije vsakega znaka in dodelitve pravemu razredu [12]. Eden od načinov klasifikacije je štetje horizontalnih in vertikalnih črt znaka. S tem zmanjšamo število vseh znakov v predlaganem razredu. Zadnji del OCR je končna obdelava. Sem spadata [12]

- grupiranje – tisti znaki, ki so bolj skupaj, verjetno pomenijo eno besedo;
- zaznavanje in popravljanje napak – glede na statistično verjetnost lahko zaznamo, ali je neka beseda pravilna; uporabimo lahko tudi slovarje.

Kljub temu pa vseeno lahko pride do napak. Za boljše rezultate obstajajo tudi načini, kjer se naprava uči na podlagi že vnaprej vnesenih znakov.

2.2 Detekcija obrazov

Detekcija obrazov ali ang. *'Face Detection'*, ki je ne smemo enačiti s prepoznavanjem obrazov ali ang. *'Face Recognition'*, je način zaznavanja obrazov na sliki/videu. Po procesiranju vhodne slike nam vrne novo sliko z označenimi obrazi.

Začne se s segmentacijo barv. To je priljubljen in zelo dober pristop k detekciji obrazov [13]. Običajno se pri slikah v formatu RGB pri spremembi svetlobne vrednosti zelo spremenijo. Zaradi tega se namesto RGB pogosto uporablja barvni prostor YCbCr. Pri detekciji barve kože se uporabljata komponenti Cb in Cr. Sledi segmentacija slike. Pri binarni sliki, ki smo jo dobili pri segmentaciji barv, se odstrani manjše bele nečistoče/šum in

(a) Segmentacija slike

(b) Označena središča na originalni sliki

Slika 2.2: Del postopka zaznavanja obrazov [13]

zapolni črne nečistoče znotraj belih prostorov. Po dodatni detekciji robov se označi središče pri vseh belih šumih (glej sliki 2.2, leva prikazuje rezultat po segmentaciji slike, desna ima označena središča v vseh belih prostorih z leve slike). Potem je treba preveriti, kateri od teh šumov so obrazi. To preverimo s t. i. lastnimi obrazi ali ang. *'Eigenface'*, ki so skupek lastnih vektorjev. Dobimo jih s testnih slik, na koncu pa izberemo najprimernejšega, ki je povprečje vseh (glej sliko 2.3). Tako dobimo čim bolj splošno obliko obraza. Pri vseh označenih pravokotnikih (ki imajo središča v prej omenjenih belih šumih) po segmentaciji slike, se nato z lastnimi obrazi preveri, ali je vsak od teh obraz. Tu pogosto pride do napak, zato jih je treba popravljati. To korigiramo npr. tako, da tiste pravokotnike, ki so preveč skupaj, združimo.

To je samo eden od načinov detekcije obrazov. Podobne pristope z lastnimi obrazi se uporablja pri slikah na socialnih omrežjih [14].

2.3 Ločevanje ključnih značilnic

Še en zanimiv pristop k ločevanju pomembnih delov slike, ki pa se ne omejuje na obraze in tekst, je ločevanje ključnih značilnic. Za razliko od podobnih algoritmov, ki temeljijo samo na svetlosti slik (sivinske slike), sem našel boljšega, ki upošteva tudi barve.

Slika 2.3: Povprečje lastnih obrazov [13]

V nadaljevanju je opisan eden od postopkov izračuna ločevanja ključnih značilnic [10]. Najprej našo vhodno sliko pretvorimo iz barvnega modela RGB v prostor XYZ, zatem pa še v prostor LMS, ki pa se še dodatno transformira. Vse to lahko predstavimo z enačbo 2.1, pri čemer je O_1 svetlost (ang. *luminance*), O_2 komponenta RG (rdeča, zelena) in O_3 komponenta BY (modra, rumena).

$$\begin{pmatrix} O_1 \\ O_2 \\ O_3 \end{pmatrix} = \begin{pmatrix} 0.3905 & 0.5499 & 0.0089 \\ -0.1764 & 0.4307 & -0.1164 \\ -0.1191 & -0.1739 & 0.8673 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{2.1}$$

Nato sledi izračun matrike M (2.2). Tu si pomagamo z enačbo 2.3, kjer je i slikovni element Gaussovega odvoda slike.

$$M = \begin{pmatrix} \overline{O_{1x}O_{1x}} & \overline{O_{1x}O_{2x}} & \overline{O_{1x}O_{3x}} \\ \overline{O_{2x}O_{1x}} & \overline{O_{2x}O_{2x}} & \overline{O_{2x}O_{3x}} \\ \overline{O_{3x}O_{1x}} & \overline{O_{3x}O_{2x}} & \overline{O_{3x}O_{3x}} \end{pmatrix} \tag{2.2}$$

$$\overline{O_{1x}O_{1x}} = \sum_{i \in I_x} (O_{1x} - \overline{O_{1x}})(O_{1x} - \overline{O_{1x}}) \tag{2.3}$$

Ko imamo izračunano matriko M, lahko na njej izvedemo razcep SVD (glej enačbo 2.4).

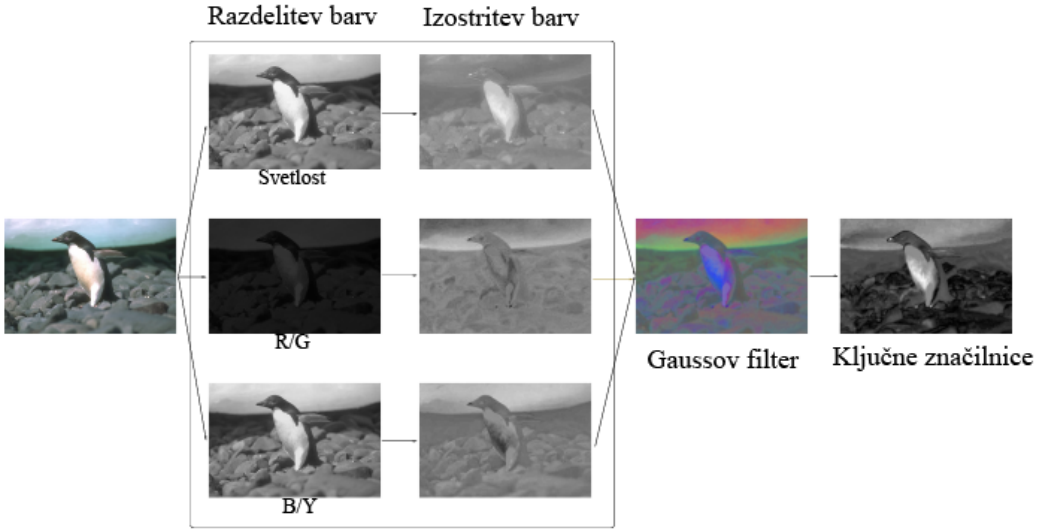
$$M = U\Sigma V \tag{2.4}$$

$$g(I_{opp}(x,y)) = U \cdot (diag(1/diag(\sqrt{\Sigma})) \cdot V^T) \cdot I_{opp} \tag{2.5}$$

Zatem moramo še izostriti barve z enačbo 2.5, kjer je I_{opp} slika, predstavljena z našimi novimi komponentami (O_1, O_2, O_3).

$$S(x,y) = ||I_\mu - I_{\omega_{hc}}(x,y)||_2 \tag{2.6}$$

Sledi še končni izračun slike, pri kateri so poudarjeni pomembnejši deli. Pri enačbi 2.6 je I_μ povprečje vrednosti, ki smo jih prej izračunali (g), $I_{\omega_{hc}}(x,y)$ pa zglajena slika z Gaussovim filtrom. Iz te razlike izračunamo evklidsko razdaljo.



Slika 2.4: Primer ločevanja ključnih značilnic

Od tod lahko vidimo, da dobimo pri veliko slikah zelo dobre rezultate, pri katerih so pomembnejši deli bolj osvetljeni. Če hočemo binarno masko, s formulo 2.7 izračunamo mejo (prag), nad katero bodo slikovni elementi 1, vsi drugi pa 0. Temu se reče upragovanje.

$$T = \frac{2}{W * H} \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} S(x, y) \tag{2.7}$$

Slika 2.5: Upragovana oz. binarna maska slike pri ločevanju ključnih značilnic

2.3.1 Upragovanje

Upragovanje je enostaven in široko uporabljen način segmentacije slike [8]. Lahko ga enostavno predstavimo s formulo 2.8. To opišemo tako, da na vhodni sliki (običajno je to sivinska slika) za vsak slikovni element preverimo, ali je večji od T. Če je, ima izhodna slika na tej poziciji vrednost 1, v nasprotnem primeru pa 0. Izhodna slika naj bi vsebovala vse pomembne oblike glavnih objektov (ločevanje ospredja in ozadja).

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \tag{2.8}$$

2.4 Rezanje šivov

K reševanju problema lahko pristopimo tudi drugače, in sicer z rezanjem šivov. Tu se dinamično spreminja velikost slike, ne da bi izgubili pomembne informacije.

Slika 2.6: Prvotna slika pri rezanju šivov

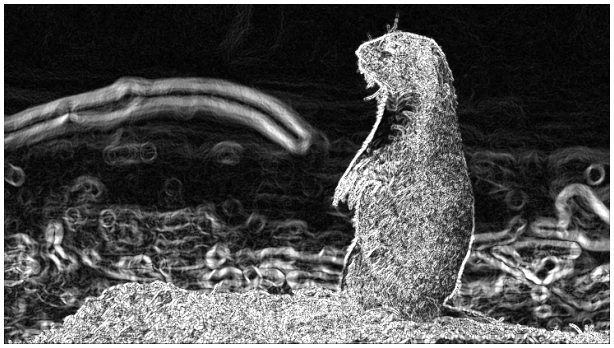
V nadaljevanju bomo govorili o tem, kaj sploh je šiv. Gre za povezano pot slikovnih elementov od zgoraj navzdol, pri čemer gre za optimalno pot [9]. Optimalnost se izračuna na podlagi energijske funkcije, ki jo izračunamo na sliki.

Algoritem deluje tako, da na vhodni sliki (2.6) najprej izračunamo energijo – tu obstaja veliko načinov in pravilnost izračuna šivov je najbolj odvisna od tega. Najboljši način za izračun energije je običajno gradient (2.7). V našem primeru je uporabljeno Scharrovo jedro, v poštev pa prideta tudi Sobelovo in Laplacovo. Ko imamo izračunano energijo, sledi izračun šivov (2.8). Tu si pomagamo z dinamičnim programiranjem. Najdemo optimalni šiv (2.9) (z najmanjšo energijo) in ga izrežemo iz slike. Ta postopek lahko ponovimo poljubnokrat oz. dokler nismo prišli do želenih dimenzij.

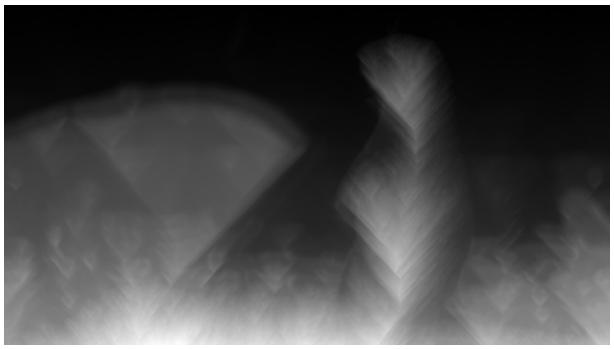
Vidimo, da je rezanje šivov zelo zanimiv način za manjšanje dimenzij slik brez izgube pomembnih objektov, vendar lahko včasih pride do manjših anomalij.

2.4.1 Gradient slike

Gradient slike je smerna sprememba intenzitete ali barve slike [2]. Uporablja se za pridobivanje informacij s slike. Gradient je diferencialna funkcija z dvema komponentama. Približke odvodov izračunamo s konvolucijo. Pri konvoluciji je najpomembnejše izbrati pravo konvolucijsko matriko oz. jedro.



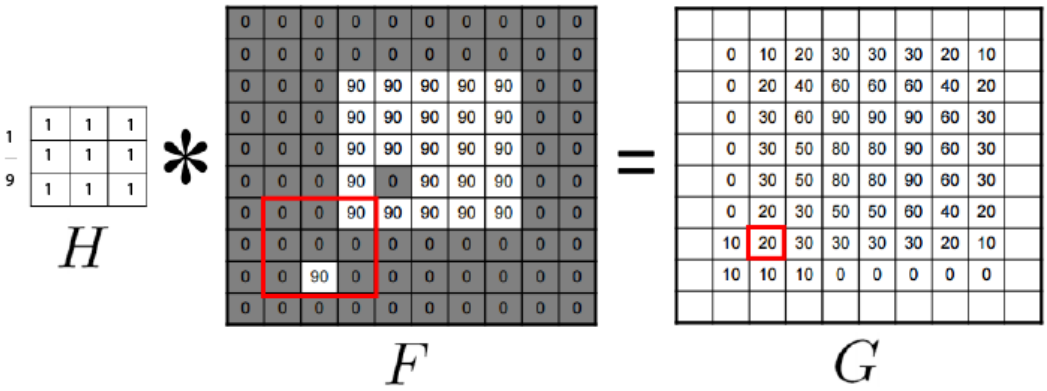
Slika 2.7: Gradient prvotne slike



Slika 2.8: Maska šivov, temnejši deli so manj pomembni

Slika 2.9: Originalna slika, na kateri je označen šiv

Slika 2.10: Izhodna slika pri rezanju šivov



Slika 2.11: Primer konvolucije [16]

Konvolucijo se lahko pri slikah realizira kot množenje slike z jedrom (ki je običajno matrika 3x3). To zapišemo z naslednjo enačbo:

$$G = H * F \tag{2.9}$$

kjer je H jedro, F pa slika [16]. Z jedrom gremo čez vsako točko na sliki, pomnožimo istoležne elemente, jih seštejemo in zapišemo v izhodno matriko. Za lažjo predstavo si pomagamo s sliko 2.11.

Različna jedra lahko uporabljamo za [3]:

- iskanje robov;
- zameglitev slike;
- ostrenje slike.

Nekatera izmed bolj znanih jeder so Gaussovo (za glajenje slike), Cannyevo, Sobelovo in Prewittovo (vsa tri za detekcijo robov). V našem primeru je pri iskanju šivov uporabljeno Scharrovo jedro, ki zgleda tako [5]:

$$G_x : \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} G_y : \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix} \tag{2.10}$$

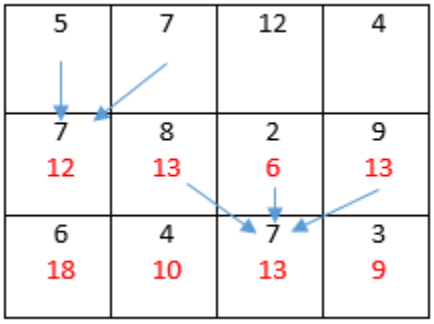
Za razliko od Sobelovega je Scharrovo bolj natančno oz. fino.

2.4.2 Dinamično programiranje

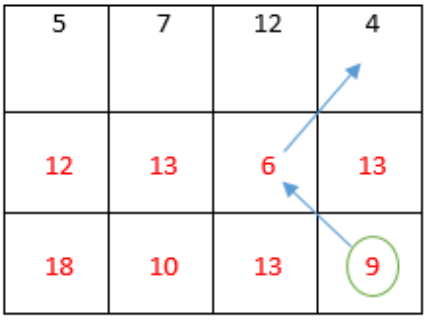
Pri dinamičnem programiranju razbijemo velik problem na manjše podprobleme. V našem primeru smo ga uporabljali za iskanje optimalnega šiva pri rezanju šivov. Izračun lahko predstavimo z enačbo 2.11 [9].

$$M(i,j) = e(i,j) + \min(M(i-1,j-1), M(i-1,j), M(i-1,j+1)) \tag{2.11}$$

To lahko ponazorimo s sliko 2.12. V praksi nastanejo slike, podobne sliki 2.8. Začnemo v drugi vrsti, kjer izmed gornjih dveh ali treh sosedov (odvisno od lokacije) izberemo najmanjšega in ga seštejemo s trenutno vrednostjo. Ko pridemo do konca, najdemo najnižjo vrednost v spodnji vrstici in sledimo najmanjšim gornjim sosedom do vrha (glej sliko 2.13). Tako smo našli optimalni šiv.



Slika 2.12: Dinamično programiranje



Slika 2.13: Iskanje optimalnega šiva

Poglavje 3

Pristop k problemu

Razvoj programa smo začeli razvijati v Matlabu, ki je programski jezik četrte generacije [18]. Zaradi same narave skriptnega jezika včasih deluje počasi. Zato smo se kasneje odločili za C++ in razvojno okolje Microsoft Visual Studio. Uporabili smo tudi knjižnico za računalniški vid – OpenCV. Program se zaganja iz ukazne vrstice, ki mora imeti določene vhodne parametre. To so:

- pot do vhodne slike;
- pot do izhodne slike;
- dimenziji izhodne slike;
- označen pomemben del – opcijsko.

Program je sestavljen iz več datotek. Ena omogoča zagon programa in preverja pravilnost vhodnih parametrov. Če so dani vsi vhodni parametri, se program nadaljuje v drugi datoteki, ki predstavlja ročno manjšanje dimenzij slik. V primeru, da imamo samo tri dane parametre (vhodna slika, izhodna slika, dimenziji izhodne slike), se program nadaljuje v tretji datoteki, ki vsebuje vse potrebno za avtomatsko manjšanje dimenzij slik.

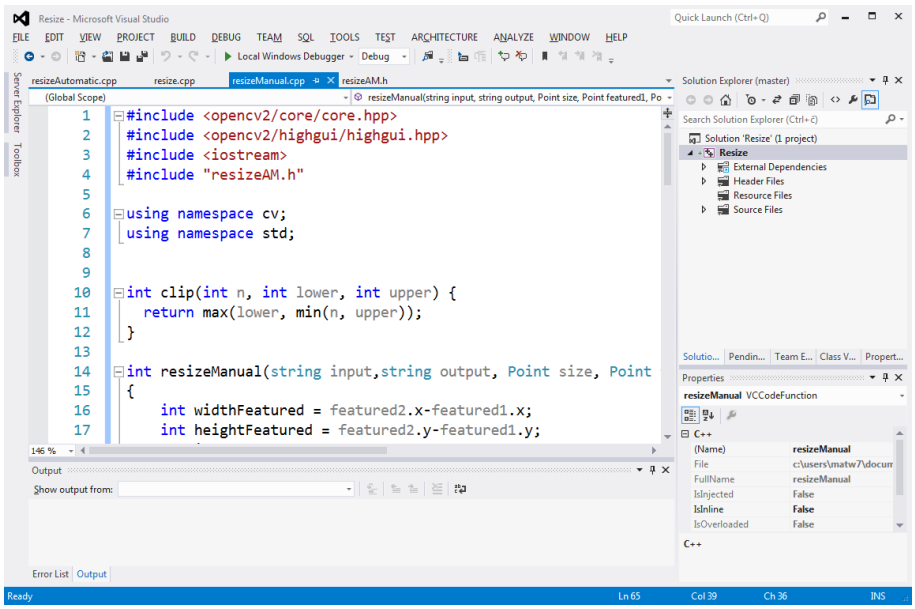
3.1 Uporabljena razvojna okolja, tehnologije in knjižnice

3.1.1 Matlab

Matlab je programski jezik oz. okolje, ki ga je razvil MathWorks. Omogoča manipulacije z matrikami, risanje funkcij, ima vključene razne algoritme in je kompatibilen s programi iz drugih programskih jezikov, kot so C, C++, Java, Fortran in Python [18]. Matlab smo uporabljali samo za izdelavo prototipa.

3.1.2 Microsoft Visual Studio

Je Microsoftovo razvojno okolje. Z njim lahko razvijamo računalniške programe, spletne strani, spletne aplikacije in storitve. Podpira naslednje programske jezike: C, C++, C#, VB.NET, F# in druge z dodatnimi nameščenimi storitvami [19]. Uporabljali smo ga za izdelavo našega programa (3.1).



Slika 3.1: Microsoft Visual Studio

3.1.3 C++

Je objektno usmerjen programski jezik, ki omogoča tudi nizkonivojsko manipulacijo s pomnilnikom. Začetki razvoja segajo v osemdeseta leta, nanj pa je vplival predvsem jezik C. Najprej je bil možen samo kot dodatek k C, standardiziran pa je bil šele leta 1998. C++ je vplival na razvoj kasnejših verzij C (C99), Java, C# in drugih [17].

3.1.4 OpenCV

OpenCV predstavlja odprtokodno knjižnico za računalniški vid. Podpira delovanje v operacijskih sistemih Windows, Linux, MacOS, iOS in Android. Z njim lahko pišemo programe v C++, C, Pythonu in Javi. Izdan je pod licenco BSD, zato se lahko uporablja tudi v komercialne namene [7]. Izmed mnogih modulov [4] smo uporabili:

- core – vsebuje osnovne funkcije in podatkovne tipe (eden najbolj uporabljenih je *Mat*) za delovanje drugih modulov;
- highgui – grafični vmesnik za prikazovanje slik, skrbi za vhodne operacije (miška, tipkovnica), branje in pisanje slik in videov na disk oz. pomnilnik;
- imgproc – manipulacije s slikami (Gaussov filter, upragovanje, gradient slike, transformacije slik ...).

3.1.5 GitHub

GitHub (GitHub.com) je spletna stran za shranjevanje kode in repozitorijev, ki temelji na Gitovem sistemu za podporo obvladovanja različic [11]. Z njim lahko enostavno prenesemo svojo kodo na repozitorij, jo popravljamo in spreminjamo. Prav tako si lahko ogledamo repozitorije drugih uporabnikov, če so javni. Z Gitom smo našo programsko kodo objavili na javnem repozitoriju na GitHubu.

3.2 Ročni izrez

Pri tem načinu moramo podati koordinate pomembnega dela slike. Koordinate so v formatu X_1xY_1-X_2xY_2, kjer X_1xY_1 predstavlja prvo točko, X_2xY_2 pa drugo. Skupaj tvorita pravokotnik. Izhodna velikost je podana na podoben način, WxH. Prva številka pomeni širino slike, druga pa višino.

Glavna ideja reševanja problema pri ročnem manjšanju dimenzij slik je izračun sredine pravokotnika, ki predstavlja pomemben del, in ustrezna razširitev višine in širine. Torej, če imamo pomemben del slike dan kot X_1xY_1-X_2xY_2, se sredina tega izračuna kot

$$(X, Y)_{sredina} = (\frac{X_1 + X_2}{2}, \frac{Y_1 + Y_2}{2}) \tag{3.1}$$

Pri OpenCV lahko v C++ iz slike izrežemo pravokotnik s pomočjo razreda *Rect()*. Pri parametrih moramo podati X in Y leve točke zgoraj ter končno višino in širino. Levo točko zgoraj dobimo tako, da prej izračunani središčni točki pravokotnika odštejemo polovico višine in širine.

$$(X, Y)_{levazgoraj} = ((X)_{sredina} - \frac{W}{2}, (Y)_{sredina} - \frac{H}{2}) \tag{3.2}$$

Zdaj imamo vse potrebne podatke za izrez iz vhodne slike. Pri izračunu točk moramo paziti na vse napake, ki se lahko zgodijo (npr. če je pomemben del na robu slike, lahko katera od točk izpade iz slike).

3.3 Avtomatski izrez

Tu mora program sam ugotoviti, kateri del je pomembnejši, in ga čim manj odrezati. To smo realizirali kot mešanico algoritmov, navedenih v prejšnjem poglavju. Na podlagi preizkušanja smo dodali še nekaj sprememb za čim hitrejše in pravilnejše delovanje.

Najprej zmanjšamo vhodno sliko. To naredimo zato, da program deluje čim manj časa v naslednjih operacijah. Sliko zmanjšamo samo v primeru, če je njena višina ali širina večja od 600 slikovnih elementov. To mejo smo

dobili s testiranjem – pri manjši meji so rezultati izreza slike slabši, pri večji pa deluje dosti počasneje.

To sliko nato vstavimo v funkcijo *calculateBySeamOrder (Mat image, int reduceRows, int reduceCols)*. Parameter *image* je vhodna slika. *ReduceRows* in *reduceCols* sta števili, ki nam povesta, koliko horizontalnih in vertikalnih šivov bomo izrezali iz slike. Pri večjem začnemo rezati šive (npr. če je *reduceCols* večji od *reduceRows*, bomo najprej odrezali *reduceCols* stolpcev slike). V nadaljevanju je del kode, s katerim iz slike izrežemo *reduceCols* vertikalnih šivov.

```
1  Mat energy = calculateEnergy(imageClone);
2
3  for(int i=0;i<reduceCols;i++)
4  {
5          Mat seamVertical=calculateSeam(energy);
6          imageClone=removeSeam(imageClone,seamVertical);
7          energy=removeSeam(energy,seamVertical);
8
9          cols=removeCol(seamVertical,cols);
10
11         if(i%20==0)
12                 energy = calculateEnergy(imageClone);
13 }
```

Iz drugega poglavja vidimo, da je treba pri rezanju šivov najprej izračunati energijo. Pri nas to naredimo s klicem *calculateEnergy(imageClone)*, kjer je *imageClone* samo kopija vhodne slike. V tej funkciji izračunamo energijo na podlagi naslednje formule, ki smo jo dobili s testiranjem

$$energy = 0.75 * ColorSaliency + 0.25 * Gradient \tag{3.3}$$

ColorSaliency je binarna maska ločevanja ključnih značilnic, ki je opisana v drugem poglavju. Gradient uporablja *Scharrovo* jedro, ki se v OpenCV

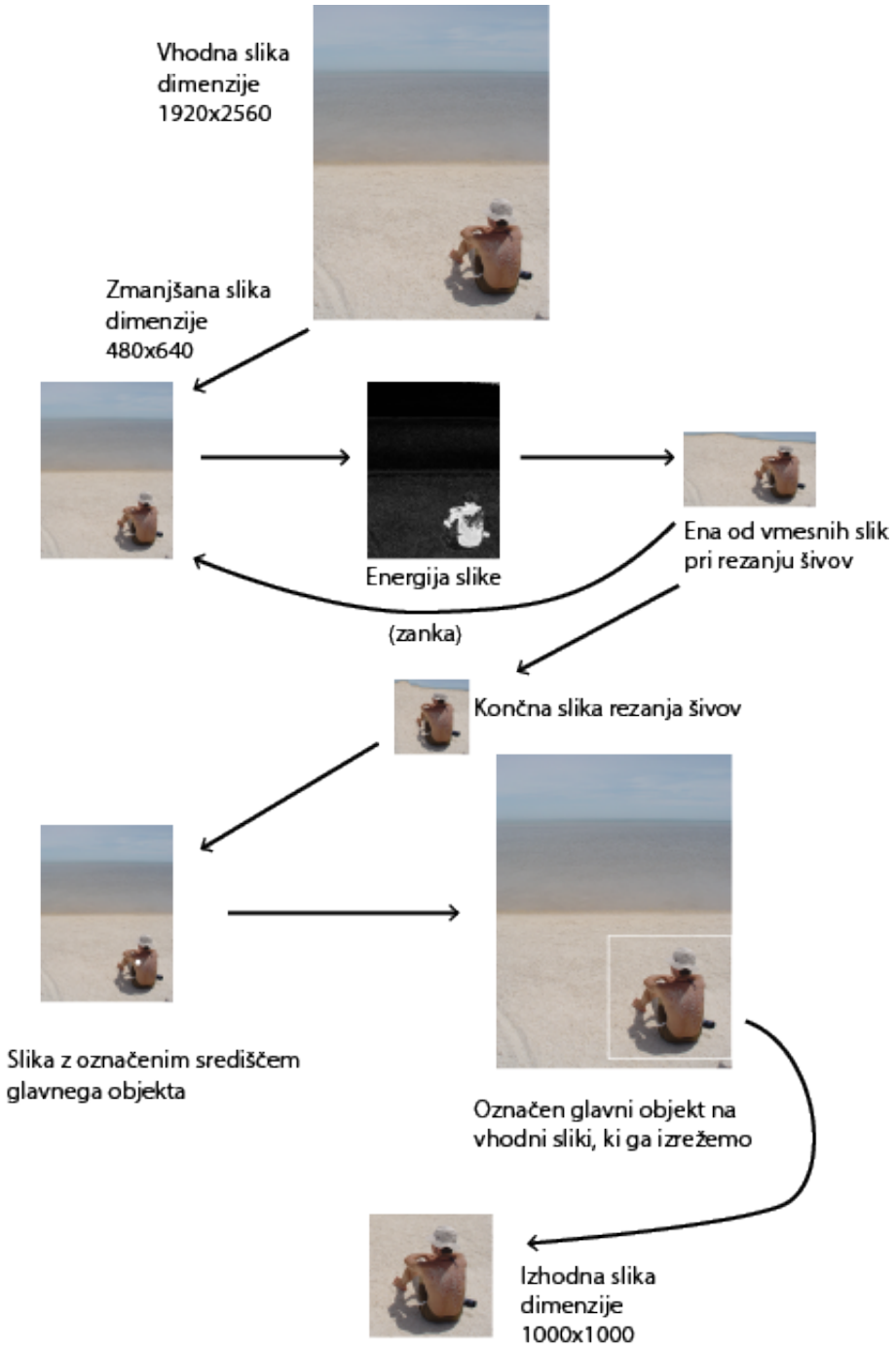
izračuna s pomočjo funkcije *Scharr()*. Celotne teže nismo dali na binarno masko, saj ta ne deluje v vseh primerih 100-odstotno.

Po izračunani energiji nadaljujemo z zanko *for*. S *calculateSeam(energy)* izračunamo masko šivov in šiv z najmanjšo energijo, ki ga funkcija tudi vrne. Ta je predstavljen kot vektor 1xN, kjer je N višina slike. Vsaka vrstica v vektorju predstavlja številko stolpca na sliki, kjer se nahaja šiv.

Nadaljujemo z odstranitvijo šiva s funkcijo *removeSeam()*. Prvi parameter določa sliko, iz katere ga odstranimo, drugi pa dejanski šiv oz. lokacija, kjer se nahaja. To odstranimo z naše delovne slike, ki jo urejamo, in iz energije slike. Zaradi optimizacije izračunamo energijo po vsakih 20 izrezanih šivih. Če bi energijo računali po vsakem izrezanem šivu, bi program deloval toliko dlje, saj sam izračun energije traja relativno dolgo. Če hočemo izrezati horizontalne šive, sliko samo rotiramo ali transponiramo.

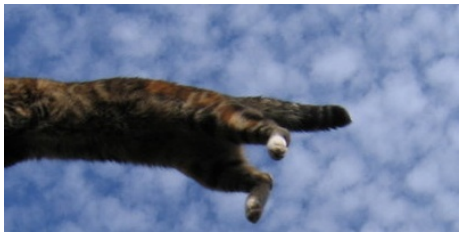
Ta način deluje dokaj dobro. Zatakne se le, če moramo izrezati veliko šivov. Takrat se slika zelo deformira in je slabo razvidna. Zato smo v naš algoritem dodali še neomenjeni spremenljivki *cols* in *rows*, s katerima računamo, kje se nahaja točka, ki je v središču slike z izrezanimi šivi, v originalni sliki. Do te ideje smo prišli tako, da smo predvidevali, da algoritem rezanja šivov potiska pomembne dele proti sredini slike. S takšno izračunano središčno točko smo potem iz originalne slike izrezali pravokotnik z ustrezno višino in širino, kjer je središče ta središčna točka. Celoten postopek si lahko lažje predstavimo vizualno s sliko 3.2.

Na sliki 3.4 je prikazan naš način testiranja in postavljanja tež pri izračunu energije (glej formulo 3.3), iz katerih smo izbrali najboljšo. Čeprav smo pri mnogo primerih dobili dobre rezultate s celotno težo na gradientu, smo pri nekaj slikah dobili še boljše z 0,75 na ključnih značilnicah in 0,25 na gradientu.

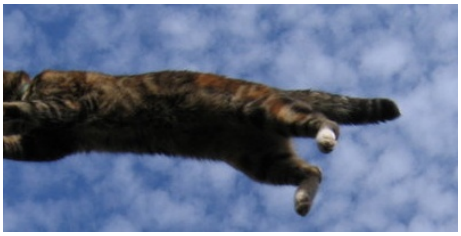


Slika 3.2: Celoten postopek avtomatskega izreza slike. Dimenzije slik niso v razmerju, temveč so namenjene le za predstavo delovanja.

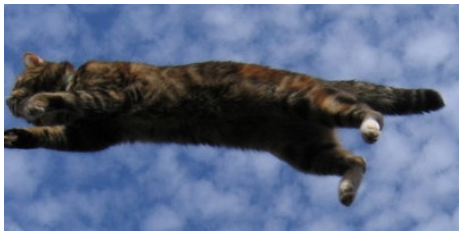
Slika 3.3: Originalna slika pred izrezom (dimenzije 900x675)



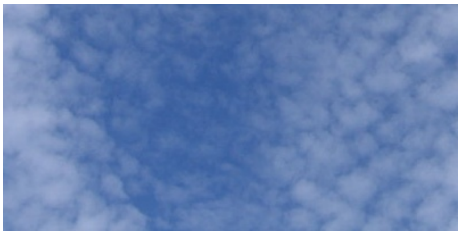
(a) *0*ColorSaliency + 1*Gradient*



(b) *0,25*ColorSaliency + 0,75*Gradient*



(c) *0,75*ColorSaliency + 0,25*Gradient*



(d) *1*ColorSaliency + 0*Gradient*

Slika 3.4: Prikaz delovanja različnih tež pri izračunu energije. Vsaka slika je dimenzije 400x200.

Poglavje 4

Rezultati

Testne slike smo izbirali med naključnimi slikami iz Googlovega iskalnika slik. V poštev so prišle tudi testne slike iz javno dostopnih baz [1]. Na primerih smo dobili različne rezultate. Eden je prikazan že v prejšnjem poglavju (glej sliko 3.2). Pri ročnem izrezovanju iz slike smo si pomagali z drugimi programi. Morali smo najti položaje točk, kjer se nahajajo pomembni deli.

Slika 4.1: Primer ročnega izreza iz slike, ki ga zaženemo tako: *Resize.exe --input cyclist.jpg --output cyclistOut.jpg --size 120x170 --featured 190x30-300x180*. Iz dimenzij 640x426 spravimo sliko na 120x170.

Ročni izrez iz slike deluje po pričakovanem, saj je enostaven za implementacijo. Težave se pojavijo pri avtomatskem izrezu, saj algoritem ne zna vedno določiti, kateri del je bolj pomemben. To je razvidno pri sliki 4.5, kjer je v izhodni sliki objekt odrezan. Še večja in bolj očitna napaka se vidi

Slika 4.2: Primer avtomatskega izreza iz slike, ki ga zaženemo z *Resize.exe --input horses.jpg --output horsesOut.jpg --size 250x100.*

Slika 4.3: Primer avtomatskega izreza: *Resize.exe --input surfer.jpg --output surferOut.jpg --size 100x100.*

na sliki 4.6. Algoritem se ne more odločiti med pomembnostjo srednjega in spodnjega dela. Po iskanju problema sem ugotovil, da se pri rezanju horizontalnih šivov veliko odreže na vrhu slike (na nebu). Zaradi rezanja na vrhu se naše središče premika proti dnu.

V drugih primerih so prikazani dobri izrezi slik.

Slika 4.4: Primer avtomatskega izreza: *Resize.exe --input horse.jpg --output horseOut.jpg --size 250x250.*

Slika 4.5: Primer avtomatskega izreza: *Resize.exe --input plane.jpg --output planeOut.jpg --size 240x140.*

Slika 4.6: Primer avtomatskega izreza: *Resize.exe --input sea.jpg --output seaOut.jpg --size 230x160.*

Poglavje 5

Pregled delovanja zaznavanja obrazov

Pri diplomski nalogi nas je še zanimalo, kako dobro se odreže algoritem za zaznavanje obrazov, implementiran v OpenCV. Pri prvem podpoglavju smo slike manjšali enakomerno in z rezanjem šivov. Pri drugem pa smo preverili detekcijo obrazov pri izgubi razmerja višine in širine slik.

5.1 Zaznavanje obrazov pri manjšanju slik

Zaznavanje obrazov smo preizkušali tako, da smo ga najprej preverili na vhodni sliki, nato pa jo postopoma zmanjševali (vsaka naslednja slika je bila za 10 % manjša od prejšnje). To smo ponovili do te točke, dokler algoritem ni več prepoznal obrazov. Na različnih primerih smo dobili podobne rezultate – zaznavanje obrazov popušča na slikah, ki so manjše od dimenzij 200x200 (številke so približne, segajo v ta rang). Seveda je veliko odvisno od velikosti slik in samih obrazov na slikah. Do tega pride zato, ker se slike pri manjšanju zameglijo in algoritem ne more več jasno določiti, kaj je obraz. V nekaterih primerih je prišlo do manjših anomalij, in sicer pri eni sliki ni bilo zaznanega obraza, pri naslednji manjši sliki pa je bil ponovno zaznan.

Detekcijo obrazov se pri OpenCV opiše z dokaj kratko kodo:

```
1  Mat grImage;
2  cvtColor(image,grImage,CV_RGB2GRAY);
3  CascadeClassifier face_cascade;
4  face_cascade.load("haarcascade_frontalface_alt2.xml");
5
6  std::vector<Rect> faces;
7  face_cascade.detectMultiScale( grImage, faces, 1.1, 2,
       0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );
8
9  // Draw circles on the detected faces
10 for( int i = 0; i < faces.size(); i++ )
11 {
12          Point center( faces[i].x + faces[i].width*0.5,
               faces[i].y + faces[i].height*0.5 );
13          ellipse( image, center, Size( faces[i].width
               *0.5, faces[i].height*0.5), 0, 0, 360,
               Scalar( 255, 255, 255 ), 4, 8, 0 );
14 }
```

Dejanska detekcija obrazov se kliče v sedmi vrstici, krogi (lahko tudi kakšne druge oblike) pa se izrišejo v zanki *for*.

(a) 481x321 (b) 186x125

Slika 5.1: Leva slika prikazuje zaznavanje obrazov pri vhodni sliki, desna pa zadnjo zaznavo obrazov po postopnem skaliranju.

(a) 321x481

(b) 139x207

Slika 5.2: Leva slika prikazuje zaznavanje obrazov pri vhodni sliki, desna pa zadnjo zaznavo obrazov po postopnem skaliranju.

(a) 481x321

(b) 284x190

(c) 186x125

Slika 5.3: Leva slika prikazuje zaznavanje obrazov pri vhodni sliki, najbolj desna pa zadnjo zaznavo obrazov po postopnem skaliranju. Na sredini je slika (anomalija), pri kateri ni bilo zaznanega obraza, čeprav je bil obraz pri naslednji zmanjšani sliki zaznan.

Na podoben način smo preverili tudi, kako deluje detekcija obrazov pri rezanju šivov. Najprej smo obraze zaznali na vhodni sliki, nato pa po naključnem številu odrezanih šivov. Logično je, da včasih pride do slabših rezultatov, saj se slika pri rezanju šivov v nekaterih primerih deformira (deformirani obrazi).

(a) 481x321

(b) 440x170

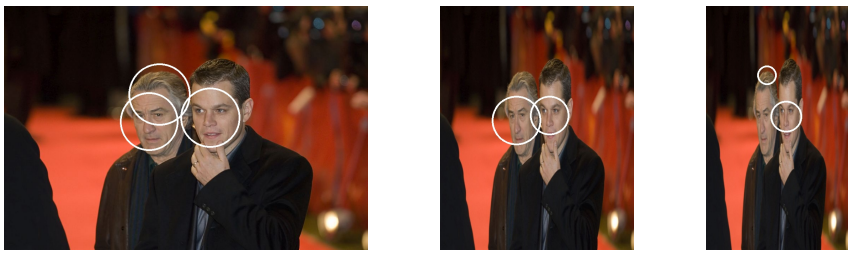
(c) 400x170

Slika 5.4: Prva slika prikazuje zaznavanje obrazov pri vhodni sliki, druga in tretja pa zaznavo obrazov po določenem številu odrezanih šivov.

5.2 Zaznavanje obrazov pri izgubi razmerja

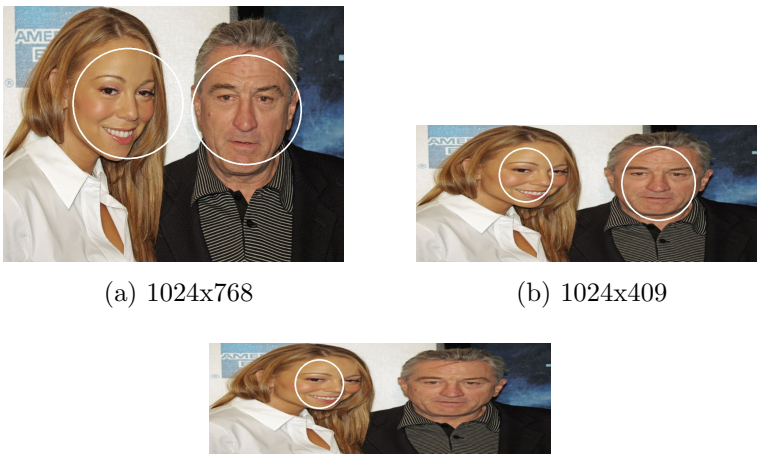
Zaznavanje obrazov pri izgubi razmerja smo preizkušali podobno kot pri prejšnjem podpoglavju, s to razliko, da slikam manjšamo samo višino ali širino (t. i. neuniformno skaliranje). S slik 5.5 in 5.6 vidimo, da pri nesorazmernem manjšanju ene od dimenzij slik tudi obrazi izgubijo svoje razmerje. Zato zaznavanje obrazov ne more določiti oblike obrazov, najti prave razdalje med različnimi deli obraza, kot so na primer razdalja med očmi, med usti in čelom in podobno. Distorzija slike torej slabo vpliva na zaznavanje obrazov.

Zadnje, kar nas je še zanimalo, je, ali izrez iz slike vpliva na zaznavo obraza. S slike 5.7, ki je izrezana iz originalne slike iz primera 5.5, vidimo, da zaznavanje obrazov pri izrezanih slikah deluje skoraj identično.



(a) 1024x685 (b) 544x685 (c) 397x685

Slika 5.5: Prva slika prikazuje zaznavanje obrazov pri vhodni sliki, druga zadnjo dokaj pravilno zaznavo obeh obrazov, tretja pa zadnji zaznan obraz pri manjšanju širine slike.



(a) 1024x768 (b) 1024x409

(c) 1024x331

Slika 5.6: Prva slika prikazuje zaznavanje obrazov pri vhodni sliki, druga zadnjo dokaj pravilno zaznavo obeh obrazov, tretja pa zadnji zaznan obraz pri manjšanju višine slike.

Slika 5.7: Izrez iz originalne slike iz primera 5.5, na kateri je opravljeno zaznavanje obrazov. Slika je velikosti 453x352.

Naše ugotovitve glede zaznavanja obrazov so, da deluje dobro, vendar le pod določenimi pogoji. Obraz mora biti zelo jasen, nedvoumen za algoritem in s pravimi razmerji med različnimi deli obraza.

Poglavje 6

Sklepne ugotovitve

V diplomski nalogi smo razvili program, ki iz vhodne slike izreže glavni objekt. Sestavljen je iz dveh delov. Prvi je ročni izrez, v katerem označimo, kateri del je pomembnejši. V drugem pa mora program sam ugotoviti, kaj naj odreže, da bo ključni objekt še vedno v izhodni sliki.

Najprej sem predelal nekaj člankov, ki se nanašajo na našo tematiko. Nato smo začeli z implementacijo algoritmov, zamenjali smo programski jezik na hitrejšega in sproti dobivali nove ideje. Temu je sledila optimizacija. Tu lahko omenimo manjšanje slike na začetku in računanje energije slike le po določenem številu izrezanih šivov. Pri diplomskem delu smo na koncu opisali še delovanje zaznavanja obrazov pod različnimi pogoji (sorazmerno/nesorazmerno manjšanje velikosti slik, po rezanju šivov, izrez iz slike), ki bi ga lahko implementirali v našem algoritmu.

Algoritem bi se dalo še bolj dodelati npr. z OCR za prepoznavanje besedila, detekcijo obrazov in podobnimi algoritmi. V poštev bi prišel tudi drugačen izračun energije. Namesto gradienta bi izračunal entropijo, ki nam pove količino informacije, ki se nahaja v slikovnem elementu. Lahko bi izbral tudi malo drugačen pristop z izračunom sopojavitvenih matrik, ki nam povedo, koliko krat se nek slikovni element pojavi v bližini drugega. Z raznimi opisniki (korelacija, kontrast, energija, homogenost ...) se potem lahko loči glavni objekt od ozadja. Kljub različnim idejam je treba vsako preizkusiti in

poiskati najboljšo.

Literatura

[1] The berkeley segmentation dataset and benchmark. Dostopno na: `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`. [Dostopano julija 2015].

[2] Image gradient. Dostopno na: `https://en.wikipedia.org/wiki/Image_gradient`. [Dostopano julija 2015].

[3] Kernel (image processing). Dostopno na: `https://en.wikipedia.org/wiki/Kernel_(image_processing)`. [Dostopano julija 2015].

[4] Opencv introduction. Dostopno na: `http://docs.opencv.org/modules/core/doc/intro.html`. [Dostopano julija 2015].

[5] Sobel derivatives. Dostopno na: `http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html`. [Dostopano julija 2015].

[6] Zemanta's programming challenge 2015. Dostopno na: `http://www.zemanta.com/zemanta-programming-challenge-2015/`. [Dostopano julija 2015].

[7] Opencv – open source computer vision. Dostopno na: `http://opencv.org`, 2015. [Dostopano julija 2015].

[8] Salem Saleh Al-Amri, Namdeo V Kalyankar, et al. Image segmentation by using threshold techniques. *arXiv preprint arXiv:1005.4020*, 2010.

[9] Shai Avidan, Ariel Shamir. Seam carving for content-aware image resizing. *ACM Transactions on graphics (TOG)*, zv. 26, str. 10. ACM, 2007.

[10] Guanqun Cao, Faouzi Alaya Cheikh. Salient region detection with opponent color boosting. *Visual Information Processing (EUVIP), 2010 2nd European Workshop on*, str. 13–18. IEEE, 2010.

[11] Laura Dabbish, Colleen Stuart, Jason Tsay, Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, str. 1277–1286. ACM, 2012.

[12] Line Eikvil. Optical character recognition. *citeseer. ist. psu. edu/142042. html*, 1993.

[13] Inseong Kim, Joon Hyung Shim, and Jinkyu Yang. Face detection. *Face Detection Project, EE368, Stanford University*, 28, 2003.

[14] Patrick Minder, Abraham Bernstein. Social network aggregation using face-recognition. *ISWC 2011 Workshop: Social Data on the Web, Bonn, Germany*, zv. 2. Citeseer, 2011.

[15] Ravina Mithe, Supriya Indalkar, and Nilam Divekar. Optical character recognition. *International Journal of Recent Technology and Engineering (IJRTE) Volume*, zv. 2, str. 72–75, 2013.

[16] Noah Snavely. Lecture 2: Image filtering. Dostopno na: `http://www.cs.cornell.edu/courses/cs6670/2011sp/lectures/lec02_filter.pdf`. [Dostopano avgusta 2015].

[17] Wikipedia. C++ – wikipedia, the free encyclopedia. Dostopno na: `https://en.wikipedia.org/wiki/C%2B%2B`, 2015. [Dostopano julija 2015].

[18] Wikipedia. Matlab – wikipedia, the free encyclopedia. Dostopno na: `https://en.wikipedia.org/wiki/MATLAB`, 2015. [Dostopano julija 2015].

[19] Wikipedia. Microsoft visual studio – wikipedia, the free encyclopedia. Dostopno na: `https://en.wikipedia.org/wiki/Microsoft_Visual_Studio`, 2015. [Dostopano julija 2015].

Priloga

Listing 6.1: Izvorna koda za ročni izrez iz slike

```
1  #include <opencv2/core/core.hpp>
2  #include <opencv2/highgui/highgui.hpp>
3  #include <iostream>
4  #include "resizeAM.h"
5
6  using namespace cv;
7  using namespace std;
8
9
10 int clip(int n, int lower, int upper) {
11   return max(lower, min(n, upper));
12 }
13
14 int resizeManual(string input,string output, Point size
       , Point featured1, Point featured2)
15 {
16         int widthFeatured = featured2.x-featured1.x;
17         int heightFeatured = featured2.y-featured1.y;
18     Mat image;
19         Mat featuredRectangle;
20         Point centerPoint;
21         Point newPoint1;
```

```
22          Point newPoint2;
23          int newWidth;
24          int newHeight;
25          Mat outputImage;
26
27
28      image = imread(input, IMREAD_COLOR); // Read the
            file
29      if(! image.data ) // Check for invalid input
30      {
31          cout << "Could not open or find the image" <<
                std::endl ;
32          return -1;
33      }
34          /*ERRORS*/
35          if( clip(size.x,0,image.size().width)!=size.x
                || clip(size.y,0,image.size().height)!=size.
                y ||
36                  clip(featured1.x,0,image.size().width)
                        !=featured1.x || clip(featured1.y,0,
                        image.size().width)!=featured1.y ||
37                  clip(featured2.x,0,image.size().width)
                        !=featured2.x || clip(featured2.y,0,
                        image.size().width)!=featured2.y ||
38                  featured1.x>=featured2.x || featured1.y
                        >=featured2.y)
39      {
40                  cout << "Wrong input" << endl;
41                  return -1;
42      }
43
```

```
44          if( size.x<widthFeatured )
45          {
46                  cout << "Warning - featured part must
                       not be cut" << endl;
47                  size.x=widthFeatured;
48          }
49          if( size.y<heightFeatured )
50          {
51                  cout << "Warning - featured part must
                       not be cut" << endl;
52                  size.y=heightFeatured;
53          }
54
55          featuredRectangle = image(Rect(featured1.x,
               featured1.y, widthFeatured, heightFeatured))
               ;
56
57          centerPoint = Point(featured1.x + widthFeatured
               /2, featured1.y + heightFeatured/2);
58
59          if( (image.cols-centerPoint.x)<size.x/2 )
               centerPoint.x=image.cols-size.x/2;
60          else if( centerPoint.x<size.x/2 ) centerPoint.x
               =size.x/2;
61
62          if( (image.rows-centerPoint.y)<size.y/2 )
               centerPoint.y=image.rows-size.y/2;
63          else if( centerPoint.y<size.y/2 ) centerPoint.y
               =size.y/2;
64
65          newPoint1 = Point(clip(centerPoint.x-size.x
```

```
            /2,0,image.size().width), clip(centerPoint.y
            -size.y/2,0,image.size().height));
66          newPoint2 = Point(clip(centerPoint.x+size.x
            /2,0,image.size().width), clip(centerPoint.y
            +size.y/2,0,image.size().height));
67          newWidth = newPoint2.x-newPoint1.x;
68          newHeight = newPoint2.y-newPoint1.y;
69          outputImage = image(Rect(newPoint1.x, newPoint1
            .y, newWidth, newHeight));
70
71          imwrite(output,outputImage);
72
73      return 0;
74  }
```

Ostala izvorna koda in celoten program sta objavljena na javno dostopnem repozitoriju na povezavi `https://github.com/matkovic/Image-resizing`.